MAIL ROOM MAR 14 1997 PAT. & TRADEMARK OFF. 63

08/820181

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application Transmittal

Assistant Commissioner for Patents
Washington, D.C. 20231
Sir:

EN995139

Transmitted herewith for filing is the **Patent Application** of:
Inventor: George William Wilhelm, Jr.
For: STATIONARY QUEUE FOR SCARCE RESOURCE MANAGEMENT
Enclosed are

[XXX] 3 sheets of drawings. (INFORMAL)

[XXX] An assignment of the invention to International Business Machines Corporation, Armonk, New York 10504,

[ ] A certified copy of a _____ application.

[XXX] Declaration and Power of Attorney

[ ] Associate Power of Attorney

[XXX] Information Disclosure Statement with PTO-1449 and copies of cited references.

The filing fee has been calculated as shown below: (Col. 1) (Col.2) Other Than Small Entity

For:	No. Filed	No. Extra	Rate	Fee
Basic Fee	////////////		////////////	$ 770.00
Total Claims	8 - 20 =	0	x $ 22.00 =	$ 0
Independent Claims	7 - 3 =	4	x $ 80.00 =	$ 320.00
[ ] Multiple Dependent Claim Presented			x $260.00 =	$ 0
			Subtotal	$
[ ] Surcharge - Late Filing Fee or Oath or Declaration			x $130.00 =	$ 0
			Total	$ 1,090.00

[XX] Please charge Deposit Account No. **09-0457** in the amount of $ 1,090.00. A duplicate copy of this sheet is enclosed.

[XX] The commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. **09-0457**. A duplicate copy of this sheet is enclosed.

[XX] Any additional filing fees required under 37 C.F.R. Sect. 1.16.

[XX] Any patent application processing fees under 37 C.F.R. Sect. 1.17.

EXPRESS MAIL CERTIFICATE
Express Mail Label No. EM 286745525 Date: 3/14/97

I hereby certify that I am depositing the enclosed or attached paper with the U.S. Postal Service "Express Mail Post Office to Addressee" service on the above date, addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.
Name of person mailing paper:: Georgia Y. Brundege
Date: 3-14-97 Signature: Georgia Y. Brundege

Respectfully submitted,
By Shelley M Beckstrand Date 3-10-97

From Shelley M Beckstrand, P.C.
Attorney at Law
314 Main Street
Owego, NY 13827
Telephone: (607) 687-9913
Fax: (607) 687-7848

a:formal1.lwp

#3

MAIL ROOM 63
MAR 11 1997
PAT. & TRADEMARK OFF.

08/820181

STATIONARY QUEUE FOR SCARCE RESOURCE MANAGEMENT

Background of the Invention

Technical Field of the Invention

This invention relates to a method and system for managing shared resources, and more particularly to a stationary queue for managing access to scarce or serially reusable resources by multiple process threads.

DOCUMENT - OCR

Background Art

Hardware device drivers that support multi-tasking operating systems must cope with the challenge of managing simultaneous access to scarce or serially re-useable resources by multiple program entities, also referred to as process threads.

When a thread requests access to a resource that is unavailable, and outright rejection of the request is not allowed, the thread must be suspended until the resource becomes available (is freed by whatever other process is currently using it). Fairness is usually important. That is, requests for the resource should be honored in the order in which they were received.

Solutions to this problem are plentiful, but involve manipulating data structures such as first in first out

(FIFOs) buffers or linked lists for managing the queuing of requests and honoring them in the order in which the occur. Such approaches are often unnecessarily complex or wasteful of either CPU cycles or memory (or both) for certain classes of this problem.

Thus, the problem of resource management has typically been solved in one of two ways: (1) free-for-all waking of any and all threads waiting for the scarce resource; or, (2) maintaining a queue structure, typically in the form of a linked list or a FIFO, to enforce order and to enable waking one thread at a time.

If multiple threads are all awakened when a single resource instance becomes available, then only one thread will be satisfied and the others will have to go to sleep (wait) again. This causes excessive churn on the system's process (or thread) run queue. Further, the free-for-all waking of all threads at once does nothing to enforce "fairness". There is, consequently, a need in the art for a method and system which is more efficiently uses system CPU resources.

In order to maintain any form of linked list structure (the classical method of queuing objects of any sort) or a first in first out (FIFO) buffer, ancillary structures must be consulted or additional memory references made in order to create the wake-up ID for the next thread in line. This requires both resources and time. There is, consequently, a need in the art for a method and system which are far faster and use less memory.

It is an object of the invention to efficiently share a scarce resource among multiple program entities (threads), and to do so with minimal CPU usage and negligible memory overhead.

It is a further object of the invention to wake waiting threads, one at time, in the exact order in which they began waiting for the resource, while minimizing system churn and guaranteeing fairness.

It is a further object of the invention to manage such resources by maintaining two counters, wherein the counters themselves not only indicate the presence (and number) of waiting threads, but contain all information necessary to wake the threads in order, one at a time.

Summary of the Invention

In accordance with the invention, a system and method are provided for managing simultaneous access to scarce or serially re-usable resources by multiple process threads comprising a stationary queue.

In accordance with the method of the invention, a sleep code routine generates a unique block identifier when a process thread temporarily cannot gain access to the resource and must be suspended; and a wakeup code routine generates a unique run identifier when a next thread in line is reanimated and granted access to the resource.

In accordance with the system of the invention, a wait counter is provided for counting the cumulative number of threads that have been temporarily denied the resource; a satisfied counter is provided for counting the cumulative number of threads that have been denied access and subsequently granted access to said resource; a sleep code routine is responsive to the wait counter for generating a run identifier; and a wakeup code routine is responsive to the satisfied counter for generating the run identifier.

Other features and advantages of this invention will become apparent from the following detailed description of the presently preferred embodiment of the invention, taken in conjunction with the accompanying drawings.

Brief Description of the Drawings

Figure 1 illustrates the programming structures for implementing the system and method of the invention.

Figure 2 is a flow diagram representation of the pseudo code statement of the method of the invention of Tables 1 and 2.

Figure 3 is a flow diagram showing the program structure of Tables 3-8 implementing the system and method of the invention.

Best Mode for Carrying Out the Invention

In accordance with this invention, the solution to this scarce resource management problem produces the same result as a queue or FIFO, but there is no memory actually associated with the queue or FIFO structure. No data or identifiers are moved into or out of this queue, hence the term 'stationary'. Only the read and write pointers of the queue need be maintained since the data that would ordinarily be stored in the queue is impliedly the counter, or pointer, values themselves.

Referring to Figure 1, to solve this problem in an efficient manner requires two brief sections of code:

1) Sleep code 120

This routine generates a unique 'Block ID' 122 when a process thread 104 temporarily cannot gain access to a scarce or serially reusable resource and must be suspended.

2) Wakeup code 130

This routine generates Run IDs 132 in the same order as the Block Ids 122 so that, when the contested resource becomes available, the next thread 102 in line is reanimated and granted access to the resource.

Multi-tasking operating systems provide system calls to perform the sleep and wakeup operations. A block command puts a thread to sleep with a unique identifier which may be

used to awaken the thread. For example, in the IBM OS/2 operating system, device drivers, or system calls, DevHlp_ProcBlock and DevHlp_ProcRun, respectively, have the capability of choosing block and run identifiers.

In addition to the code fragments 120, 130, two companion counter variables 110, 112 are required as defined below. These counters may be any length (8-bit, 16-bit, etc.) large enough to accommodate the maximum number of outstanding queued requests for the resource in question, and they must be the same length.

1) number_forced_to_wait (NFW) 112

This counter 112 is a cumulative counter of the number of threads that have been temporarily denied the resource. The initial value of this counter is zero (0). It is incremented whenever a thread 102 is denied access to the resource and is forced to wait or sleep until it becomes available. When this counter is incremented past its maximum value, it rolls over to zero.

2) number_satisfied (NSC) 110

This counter 110 is a cumulative counter of the number of threads 102 that have been first forced to wait and have been subsequently satisfied in their demand to access the scarce resource. The initial value of this counter is zero (0). Whenever a thread exits the head of the line 100 and is granted access to the resource, NSC counter 110 is

incremented. When NSC counter 110 is incremented past its maximum value, it rolls over to zero.

Whenever the number_forced_to_wait (NFW) counter 112 is exactly equal to the number_satisfied (NSC) 110 counter, then there are no threads 100 waiting for the resource, and no wakeup calls must be made. However, when these counters 110, 112 are not equal, then there are threads 102, 104 waiting in the stationary queue 110, 112, 114 for access to the scarce resource, and a wakeup call must be issued whenever a resource becomes available. The queuing (sleeping) and dequeuing (awakening) operations are depicted in the following sections.

In both sections (below), reference is made to creating a Block ID 122 or Run ID 132 using one of the two stationary queue counters 110, 112. This refers to creating a number of the proper length for use by the Sleep (Block) function 120 or Wakeup (Run) function 130 for the target operating system. For example, in the OS/2 operating system, Block/Run Ids 122, 132 are 32 bit values. Further, they must be distinct from those used for other purposes of the same device driver or by other device drivers in the system. To accomplish the latter requirement, OS/2 device drivers typically use their own code or data addresses as the base 114 for forming a Block/Run ID 122, 132. Thus, in the program flows below, 'Create a Block ID' refers to the process of adding the appropriate counter's 110, 112 value to a given base number 114, the selection of which is not germane to this invention.

Referring to Figure 2, the procedure for adding a thread 102 to the stationary queue is set forth in Table 1, and the procedure for removing a thread 104 from the stationary queue is set forth in Table 2. In step 140, number_forced_to_wait counter (NFW) 112 and number_satisfied counter (NSC) 110 are initialized to zero. Upon receiving a request for a resource in step 142, in step 144 the determination is made if the resource is available. If the resource is not available, in step 146 a block ID 122 is created using NFW 112, NFW 112 is incremented in step 148, and the thread is blocked in step 150 to await a wake-up call at step 162, in which the run ID 132 is equal to the block ID with which it was put to sleep in step 150.

If, in step 144, it is determined that the requested resource is available, in step 145 the thread uses the resource and, when finished, in step 152 returns the resource. In step 154 it is determined whether or not NFW counter 112 equals NSC cunter 110. If they are equal, there are no threads waiting on the resource, and processing returns to step 142 to await a request from a thread for the resource. If they are not equal, there is a waiting thread. In step 156, run ID 132 is created using the value of NSC 110 counter, which will then be incremented in step 158. In step 160, the thread identified by run ID 132 is awakened and made ready to run by the operating system. In step 162, that thread awakens, and in step 145, uses the resource.

Referring to Figure 3 in connection with Tables 3-8, a more detailed description of a preferred embodiment of the invention is set forth.

Table 1: Add Thread

```
IF the resource is unavailable THEN DO

  Create a Block ID using the number_forced_to_wait counter
  Increment the number_forced_to_wait counter
  Block this thread using the Block ID just created

     /* At this point this thread of execution is       */
     /* suspended, waiting for its unique wake-up call. */

     /* When wake-up call comes, execution resumes at   */
     /* this point.                                     */

END
```

The resource is now available, and the thread commences using it.

Table 2 Remove Thread

```
IF number_forced_to_wait is NOT EQUAL TO number_satisfied
  THEN DO

     Create a Run ID using the number-satisfied counter
     Increment the number-satisfied counter
     Run the thread using the Run ID just created

     /* At this point, the process thread that has been */
     /* waiting in the stationary queue the longest is  */
     /* made ready to run by the operating system.      */

END
```

Table 3: Component Header

```
/
 *  COMPONENT NAME:  statq.c
 *
 *     DESCRIPTION:  Generic function for implementing a
 *     delicatessen-style queuing mechanism - lightweight and
 *     fast.
 *
 *     REFERENCES:  Physical Device Driver Reference for OS/2
 *     (S10G-6266-01)
 *
 *       FUNCTIONS:  InitStationaryQueue()
 *       InitAllStationaryQueues()
 *       NowServing()
 *       GetInLine()
 *
 *
 *  ORIGINS:
 *
 *  (C) COPYRIGHT International Business Machines Corp., 1995
 *  All Rights Reserved
 *  Licensed Materials - Property of IBM
 *
 */
```

Confidential - Property of Symbios

Table 4: Include/Import/Local Macros

```
/*
 *  This structure defines a "stationary queue".
 *  This queue is a lightweight, delicatessen-style, serialization
 *    mechanism that enforces orderly access to a scarce resource.
 *  Threads requesting access to a resource "take a number".
 *  As their numbers are called, the threads are awakened.
 *  The base value of the queue is a unique virtual address in
 *    our data storage.
 *  Multiple queues must have unique base values separated by at
 *    least the queue size (which is 256).
 *  The queue can hold at most 255 customers.  No checking is
 *    performed to enforce this in this embodiment.
 */

struct STATIONARY_Q {
   ULONG base;       /* base value for forming Block/Run Ids       */
   UCHAR next;       /* aka number_satisfied_counter, or NSC       */
   UCHAR last;       /* aka number_forced_to_wait counter, or NFW */
};

 /*
  *    IMPORTED FUNCTIONS/DATA:
  */
extern struct ACB;
#endif                    /* _AIX                                  */

/*
 *    LOCAL MACROS:
 */
#define  QUEUE_SIZE  256       /* default max size                  */
#define  STATIONARY_Q_BASE ((ULONG)((char far *) &acb))
                           /* arbitrary q base guaranteed to be
                                in our data seq                     */
```

Table 5: InitStationaryQueue()

```
/*
 *    InitStationaryQueue()
 *
 *    This function initializes a single stationary queue
 *      structure.
 *    If the given base is non-zero, it is used.
 *    Otherwise, the base used is the virtual address of the ACB
 *      structure.
 *    The base used is incremented by the QUEUE_SIZE and returned
 *      to the caller for use on subsequent calls.
 *    It is anticipated that this routine will be called multiple
 *      times to initialize multiple queues (the design assumption
 *      here is that a queue of size 256 suffices).
 */

ULONG
M_InitStationaryQueue
   (
   AIX_ACB_PARM          /* Parameter for AIX only - DO NOT follow
                            with a comma                          */
   int   queue_number,
   ULONG base
   )
{
   struct STATIONARY_Q *pq;

   pq = &acb.stationary_q[queue_number];

   if (base == 0L)
      base = STATIONARY_Q_BASE;

   pq->base = base;

   pq->next =
   pq->last = 0;

   base += QUEUE_SIZE;

   return( base );
}                        /* end of InitStationaryQueue()          */
```

Table 6: InitAllStationaryQueues()

```
/*
 *    InitAllStationaryQueues()
 *
 *    This function initializes all stationary queue objects for
 *      the DD.
 *    The assumption here is that they *all* are the same size.
 *    The q_base argument to InitStationaryQueue is set to ZERO the
 *      first time through to indicate that this is the first queue
 *      initialized.
 *    Thereafter, the previous return value from
 *      InitStationaryQueue is fed back to cause the queue ranges
 *      to be sequential and not overlapping.
 */

void
M_InitAllStationaryQueues
  (
  AIX_VOID_PARM
  )
{
   ULONG q_base;
   int   q;


   q_base = 0L;

   for ( q = 0 ; q < NUM_QUEUES ; ++q )
      q_base = InitStationaryQueue( q, q_base );

   return;
}                        /* end of InitAllStationaryQueues()       */
```

Table 7: NowServing()

```
/*
 *    NowServing()
 *
 *    This function calls the next number of a thread sleeping in
 *      the stationary queue.
 *    A function relinquishing a scarce resource will call this
 *      routine.
 *    Since some threads may have awakened prematurely (timed out)
 *      and may no longer be holding a number, this function should
 *      continue to call outstanding numbers until some thread
 *      responds or the queue is empty.
 */

void
M_NowServing
   (
   AIX_ACB_PARM          /* Parameter for AIX only - DO NOT follow
                            with a comma                          */
   int   queue_number
   )
{
   struct STATIONARY_Q *pq;  /* pointer to the queue of interest */
   int      num_threads_woke;

   pq = &acb.stationary_q[queue_number];

   /*
    *  While there are numbers left to call, call the next in line
    *    until someone answers or the queue is emptied.
    */

   while (pq->next != pq->last) {

      num_threads_woke = Run( ++pq->next + pq->base );

      if (num_threads_woke > 0)
         break;
   }
   return;
}                        /* end of NowServing()                  */
```

Table 8: GetInLine()

```
/*
 *    GetInLine()
 *
 *    This routine places a thread in the stationary queue.
 *    If a thread wants a scarce resource that is not available, it
 *      calls this routine to get in line for the resource.
 *    Basically, the thread uses this function to "take a number".
 *    When the number is called, this thread awakens.
 *
 *    It is assumed that interrupts are DISABLED before getting
 *      here so that the wake-up call (Run) does not precede our
 *      wait (Block).
 *    Interrupts are re-DISABLED upon return from Block.
 *
 *    The return value from Block is passed back out to the caller.
 */

int
M_GetInLine
   (
   AIX_ACB_PARM          /* Parameter for AIX only - DO NOT follow
                            with a comma                          */
   int   queue_number
   )
{
   struct STATIONARY_Q *pq;  /* pointer to the queue of interest */
   int      rc;

   pq = &acb.stationary_q[queue_number];   /* access the desired
                                                 queue           */

   /*
    *  Get in at the end of the line.  Bump the last counter (take
    *    a number) and sleep on the number taken.
    */

   rc = Block( ++pq->last + pq->base, -1, INTERRUPTIBLE );

   return( rc );
}                        /* end of GetInLine()                   */
                         /* end of statq.c                       */
```

Advantages over the Prior Art

It is an advantage of the invention that it provides an apparatus and method applicable to any device driver design that serves multiple processes, and to application programs that utilize multiple threads accessing common (shared) scarce resources, such as is the case with device drivers or application programs for multi-tasking operating system, such as OS/2, AIX, UNIX, and Windows 95.

The problem of resource management has typically been solved in one of two ways: (1) free-for-all waking of any and all threads waiting for the scarce resource; or, (2) maintaining a queue structure, typically in the form of a linked list or a FIFO, to enforce order and to enable waking one thread at a time.

The advantage provided by the invention with respect to the first alternative listed above is that it more efficiently uses system CPU resources. If multiple threads are all awakened when a single resource instance becomes available, then only one thread will be satisfied and the others will have to go to sleep (wait) again. This causes excessive churn on the system's process (or thread) run queue. Further, the free-for-all waking of all threads at once does nothing to enforce "fairness". The method and apparatus of the invention wakes waiting threads, one at time, in the exact order in which they began waiting for the resource, minimizing system churn and guaranteeing fairness.

The advantage provided by the invention with respect to the second alternative listed above is that it is far simpler to maintain two trivial counters than to maintain any form of linked list structure (the classical method of queuing objects of any sort) or a first in first out (FIFO) buffer. The counters themselves not only indicate the presence (and number) of waiting threads, but contain all information necessary to wake the threads in order, one at a time. No ancillary structures need be consulted nor additional memory references made in order to create the wake-up ID for the next thread in line. Thus, the method and apparatus of the invention are far faster and use less memory.

Alternative Embodiments

It will be appreciated that, although specific embodiments of the invention have been described herein for purposes of illustration, various modifications may be made without departing from the spirit and scope of the invention. In particular, it is within the scope of the invention to provide a memory device, such as a transmission medium, magnetic or optical tape or disc, or the like, for storing signals for controlling the operation of a computer according to the method of the invention and/or to structure its components in accordance with the system of the invention.

Accordingly, the scope of protection of this invention is limited only by the following claims and their equivalents.

Document Page 22 of 23

CLAIMS

We claim:

1. A multi-tasking operating system for managing simultaneous access to scarce or serially re-usable resources by multiple process threads, comprising:

at least one resource;

a plurality of threads; and

a stationary queue for allocating access to said resource amonst said threads.

2. A multi-tasking operating system stationary queue for managing simultaneous access to scarce or serially re-usable resources by multiple process threads, the stationary queue comprising:

a sleep code routine for generating a unique block identifier when a process thread temporarily cannot gain access to said resource and must be suspended; and

a wakeup code routine for generating a unique run identifier when a next thread in line is to be re-animated and granted access to said resource.

3. The system of claim 2, further comprising:

a wait counter for counting the cumulative number of threads that have been temporarily denied the resource;

a satisfied counter for counting the cumulative number of threads that have been denied access and subsequently granted access to said resource;

said sleep code routine being responsive to said wait counter for generating said run identifier; and

said wakeup code routine being responsive to said satisfied counter for generating said run identifier.

4. A method for managing simultaneous access to scarce or serially re-usable resources by multiple process threads, comprising the steps of:

responsive to a request for a resource which is not available, creating a block identifier based on the number of threads temporarily denied the resource; and

blocking the thread using said block identifier.

5. A method for managing simultaneous access to scarce or serially re-usable resources by multiple process threads, comprising the steps of:

responsive to a resource becoming available, creating a run identifier based on the number of threads that have been first forced to wait and have been subsequently satisfied; and

running the thread using said run identifier.

Docket EN995139

6. A method for managing simultaneous access to scarce or serially re-usable resources by multiple process threads, comprising the steps of:

responsive to a request for a resource which is not available,

creating a block identifier based on the number of threads temporarily denied the resource; and

blocking the thread using said block identifier; and

responsive to a resource becoming available,

creating a run identifier based on the number of threads that have been first forced to wait and have been subsequently satisfied; and

running the thread using said run identifier.

7. A memory device for storing signals for controlling the operation of a computer to manage simultaneous access to scarce or serially re-usable resources by multiple process threads, according to the steps of

responsive to a request for a resource which is not available,

creating a block identifier based on the number of threads temporarily denied the resource; and

blocking the thread using said block identifier; and

responsive to a resource becoming available,

creating a run identifier based on the number of threads that have been first forced to wait and have been subsequently satisfied; and

running the thread using said run identifier.

8. A memory device for storing signals to structure the components of a digital computer to form a stationary queue for managing simultaneous access to scarce or serially re-usable resources by multiple process threads, comprising:

a sleep code routine for generating a unique block identifier when a process thread temporarily cannot gain access to said resource and must be suspended;

a wakeup code routine for generating a unique run identifier when a next thread in line is to be re-animated and granted access to said resource;

a wait counter for counting the cumulative number of threads that have been temporarily denied the resource;

a satisfied counter for counting the cumulative number of threads that have been denied access and subsequently granted access to said resource;

said sleep code routine being responsive to said wait counter for generating said run identifier; and

said wakeup code routine being responsive to said satisfied counter for generating said run identifier.

STATIONARY QUEUE FOR SCARCE RESOURCE MANAGEMENT

Abstract of the Disclosure

A system and method for managing simultaneous access to a scarce or serially re-usable resource by multiple process threads. A stationary queue is provided, including a wait counter for counting the cumulative number of threads that have been temporarily denied the resource; a satisfied counter for counting the cumulative number of threads that have been denied access and subsequently granted access to said resource; a sleep code routine responsive to the wait counter for generating a run identifier; and a wakeup code routine responsive to the satisfied counter for generating the run identifier.

Docket - Search

08/820181

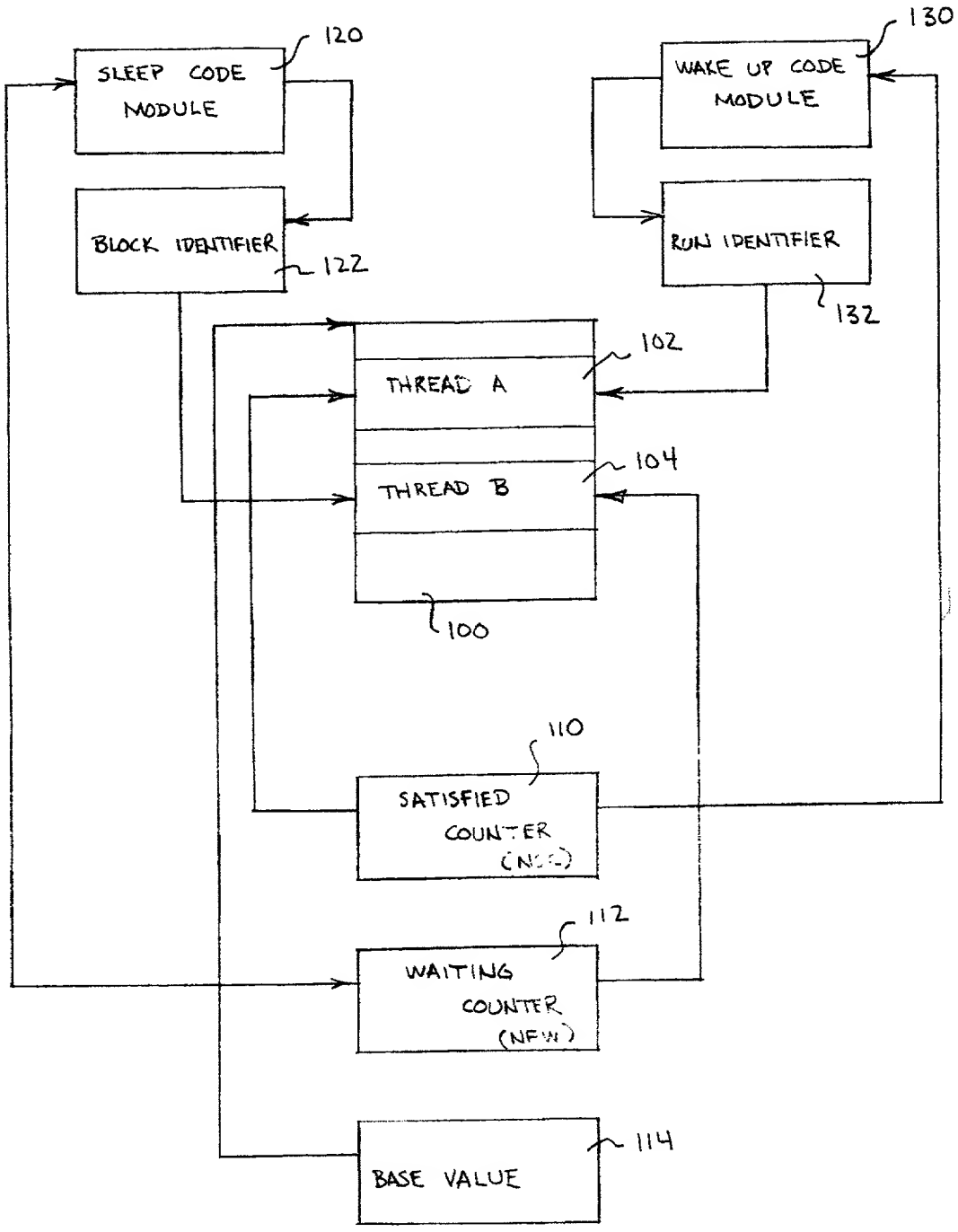

FIGURE 1

08/820181

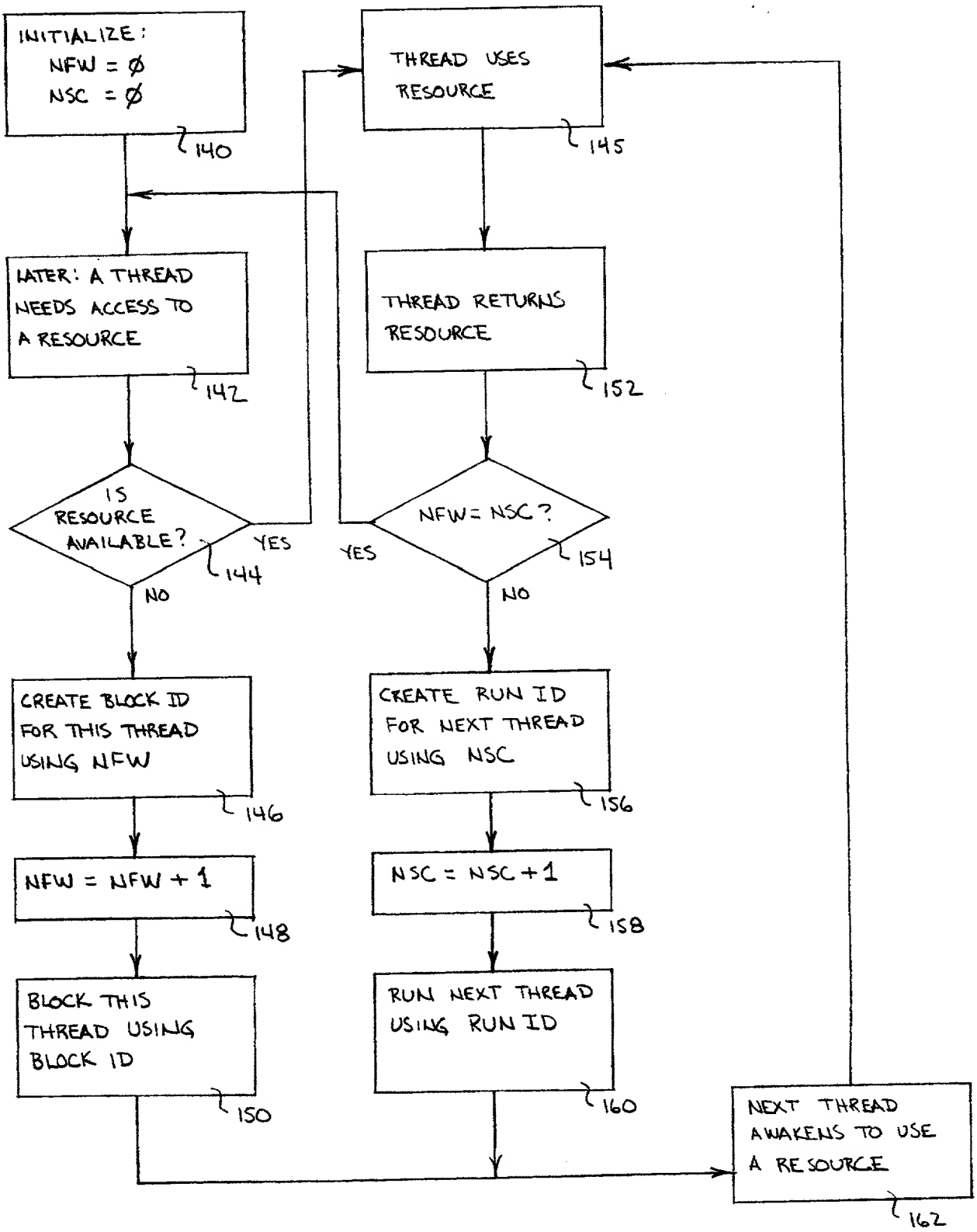

FIGURE 2

08/820181

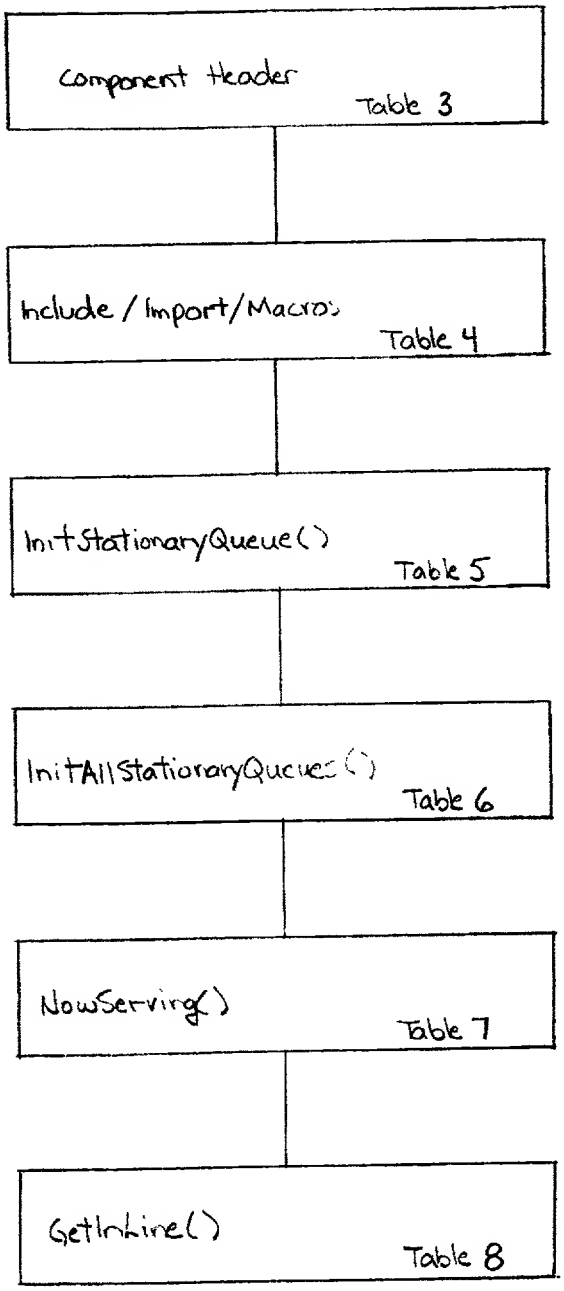

FIGURE 3

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

Atty.Docket No. EN995139

Express Mail #EM286745525

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name; I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:
STATIONARY QUEUE FOR SCARCE RESOURCE MANAGEMENT

the specification of which (check one)

___XXX___ is attached hereto.

_____ was filed on _____ as Application Serial No. _____ and was amended on _____.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Reualtions, section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, section 119 of any foreign application(s) for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Appplication(s):

Number	Country	Daty/Month/Year	Priority Claimed

I hereby claim the benefit under Title 35, United States Code, section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States applications in the manner provided by the first paragraph of Title 35, United States Code, section 112, I acknowledge the duty to disclose information material to the patentability of this application as defined in Title 37, Code of Federal Regulations, section 1.56 which occurred between the filing date of the prior application and the national or PCAT international filing date of this application:

Prior U.S. Applications:

Serial No.	Filing Date	Status

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

As a named inventor, I hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith: David L. Adour, Reg. No. 29,604; Lawrence R. Fraley, Reg. No. 26,885; Richard M. Goldman, Reg. No. 25,585; Arthur J. Samodovitz, Reg. No. 31,297; Bernard Tiegerman, Reg. No. 29,707; Christopher A. Hughes, Reg. No. 26,914; Edward A. Pennington, Reg. No. 32,588; John H. Hoel, Reg. No. 26,279; Joseph C. Redmond, Jr., Reg. No. 18,753; and Shelley M Beckstrand, Reg. No. 24,886.

Send all correspondence to: Shelley M Beckstrand, P.C.
Attorney at Law
314 Main Street
Owego, NY 13827

Direct telephone calls to: (607) 687-9913 [alt: (607) 755-3268]

Inventor: George William Wilhelm, Jr.

Signature: George William Wilhelm Jr. Date: 3/14/97

Residence: 705 Catalina Boulevard, Endwell, New York 13760

Citizenship: USA

Post Office Address: Same

a:formal7.lwp